Computer Lab 1

Clara Escorihuela Altaba
claraea@kth.se
19980504-T283

Joana Palés i Huix
joanaph@kth.se
19970213-4629

1 Problem 1: The Maze and the Random Minotaur

1.1 Markov Decision Process formulation

- **State space:** all the combinations of possible positions of the player (i,j) and the minotaur (k,l) in the maze, excluding impossible states, i.e. where there are the walls of the maze.
  $S = \{(i,j,k,l)$: such that the neither the cell (i,j) nor (k,l) is an obstacle$\} \cap S_{terminalstate} = \{(i,j) == (6,5) \cup (i,j) == (k,l)\}$
- **Action space:** all the actions that the player can choose.
  $A = \{$stay, up, down, left, right$\}$
- **Transition probabilities:** since the movement of the player and the movement of the minotaur are independent, the probability of the state given an action will be the multiplication of probabilities. That is,
  - **Player position:** depends on the action $P((i',j') \mid (i,j),a) = 1$ if (i',j') is not a wall or obstacle. Otherwise, $P(i',j' \mid (i,j),a) = 0$.
  - **Minotaur position:** $P(k',l' \mid (k,l)) = 1/n$ where $n \in A_p$, being A_p the number of possible actions due to the current minotaur position, and (k',l') is not a wall. Otherwise, $P(k',l' \mid (k,l)) = 0$.

  Therefore,
  - $P((i',j',k',l') \mid (i,j,k,l),a) = P((i',j') \mid (i,j),a)P((k',l') \mid (k,l)) \quad \forall$ (i,j) $\neq (k,l)$
  - $P((i',j',k',l') \mid (i,j,k,l),a) = 0 \quad \text{if} \quad (i,j) == (k,l)$
  - $P((i,j,k,l) \mid (i,j,k,l),a) = 1 \quad \text{if} \quad (i,j) == (k,l)$
  - $P((i',j',k',l') \mid (i,j,k,l),a) = 0 \quad \text{if} \quad (i,j) == (x_{win},y_{win})$
  - $P((i,j,k,l) \mid (i,j,k,l),a) = 1 \quad \text{if} \quad (i,j) == (x_{win},y_{win})$

1.2 Dynamic Programming

Dynamic programming aims at estimating the state value function of a given policy by backward induction. The value function corresponds to the state value of the optimal policy (taking into account we know the transition probabilities and the reward function, that we have a terminal state and a finite number of policies): $V_t^*(s) = max_{\pi \in \Pi} V_T^{\pi}(s)$.

Therefore, we obtain $V_T^*(s) = u_1^*(s)$ for all $s \in S$, and we can find the optimal police as 1.

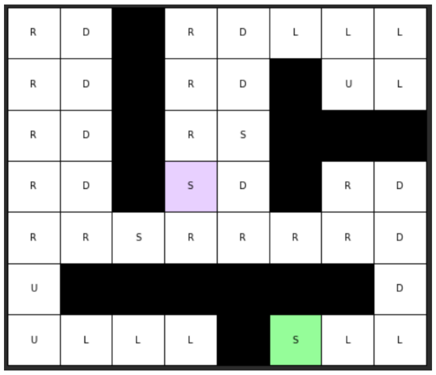
$$\pi_t^*(s) \in arg\ u_1^*(s) \tag{1}$$

$$\text{where } u_1^*(s) = \max_{a \in A_s} Q_t(s,a) \tag{2}$$

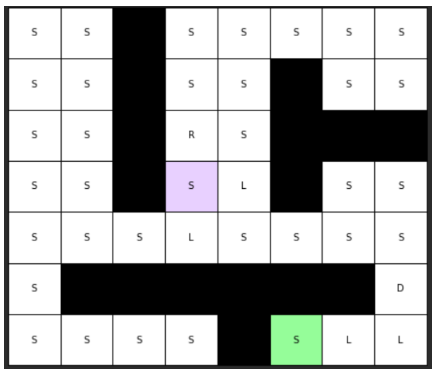
$$\text{where } Q_t(s,a) = r_{t-1}(s,a) + [\sum_{j \in S} p_{t-1}(j|s,a)u - t^*(j) \tag{3}$$

In this case, we have estimated the best action the player should do in a given position taking into account the current reward and the estimation of all possible future rewards. Figures 1 show the policy (for a 20 step time horizon and an initial Minotaur position equal to (3,3)) at time 4 and 17 when the Minotaur can and cannot stay in a position. From these results we can concluded that policy is time dependent, due to it varies in respect to the number of future events, and consequently, the time step.

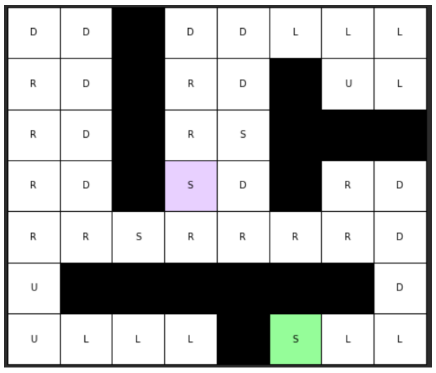
Furthermore, the possible movement of the Minotaur affects the transition probabilities and consequently the value function, reason why the policies vary in the same time (position (0,0) in figure 1a and 1c). Another interesting fact is that at time 17 most of the actions correspond to stay, because the the player will not have enough time to arrive to the end, and consequently staying in the same position has a lower penalty than moving towards the end.



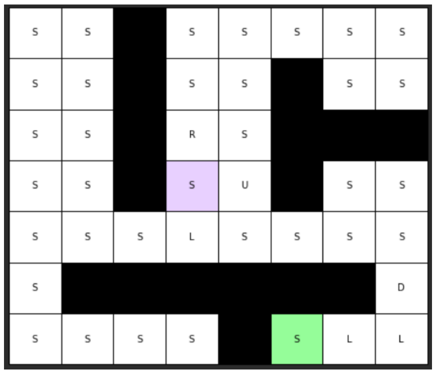
(a) t=4, staying=False



(b) t=17, staying=False



(c) t=4, staying=True



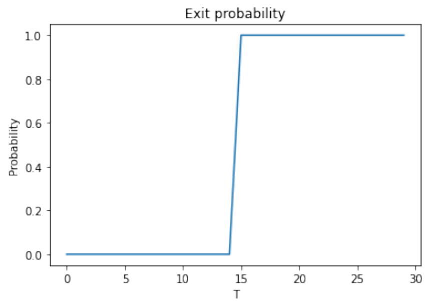
(d) t=17, staying=True

Figure 1: Policies

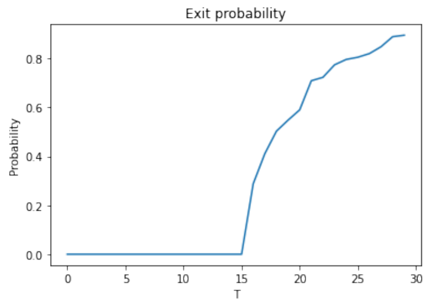
Figures 2 shows the exit probability of the player for an increasing time horizon when the Minotaur cannot and can stay. Both cases result in an exit probability equal to 0 when the time horizon is lower than 12, because the distance between the first and last box is higher than 15, and consequently the player does not have enough time to arrive to the end.

As expected, over 15 steps the probability the player has to win is equal to one if the minotaur cannot stay in his position. The reason is because when the player is located next to the minotaur he will jump to the minotaur box, because he will not be able to stay there, and the player will be safe. Therefore, the only way of being killed is when there is one box between the minotaur and the player and both of them jump towards this. However, the probability of this to happen is extremely low, for that reason the exit probability is equal to one.

This fact is different when the minotaur can stay, because the player can not be sure whether he will be saved while jumping towards the current position of the minotaur, consequently the probability of exiting decreases.



(a) Minotaur can't stay



(b) Minotaur can stay

Figure 2: Exit probabilities with different time horizons

1.3 Value Iteration

Subsequently, the player has been poisened and needs to leave the game as soon as possible. We have simulate this enviroment as a infinite time horizon MDP where the player has a random geometrical distributed time horizon with mean equal to 30.

In this case, V^* is the unique solution of the Bellman's equation, and:

$$\pi_t^*(s) \in arg\ V^*(s) \tag{4}$$

$$\text{where } V^*(s) = \max_{a \in A_s} Q_t(s,a) \tag{5}$$

$$\text{and } Q(s,a) = r(s,a) + \lambda \sum_{j \in S} p(j|s,a) max_{b \in A_j} Q(j,b) \tag{6}$$

The time horizon (T) has been obtain by selecting a random sample of a geometrical distribution and the *k* parameter has been set to 29/30 because of eq.7. That means that the policy has been calculated until the value iteration algorithm has converged ($\epsilon < 0.0001$) or the player has stopped playing because he has arrived to the final position or to the minotaur position or due to the venom has killed him (end of the time horizon).

$$\mathbb{E}[T] = \mu = 1/(1-\lambda) \tag{7}$$

Based on these conditions we have estimated the exit probability of the runner over 10000 times, which has derived in 61% and 51% while the minotaur can and cannot stay respectively. It is interesting to highlight that both values are lower than in the finite MDP case, because of the time horizon randomization.

1.4 Q-learning and Sarsa

1.4.1 Theoretical questions

- **What does it mean that a learning method is on/off policy?**
  It refers to whether the agent learns the value of the policy under which the data is generated or not. Online policy learning is an active process, due to the policy is updated based on the data collected. Therefore, in online policy the behaviour and the update policy is the same. On the other hand, in offline policy the behaviour policy stays the same all time and it is not updated based on the observations of the agent, therefore, the player always follows the behaviour policy to decide the next action. Q learning is an off policy algorithm while SARSA is an online one.
- **State the conditions for Q learning and SARSA**
  The conditions for the Q-learning algorithm to converge are:
  - $\sum_t \alpha_t = \inf$ and $\sum_t \alpha_t^2 < \inf$

– The behaviour policy π_b visits each pair of (state,action) infinetily often

Which are met if $\alpha_t = \frac{1}{t+1}$ and the π_b yields an irreducible Markov Chain

The conditions for the SARSA algorithm to converge are:

– $\sum_t \alpha_t = \inf$ and $\sum_t \alpha_t^2 < \inf$

– The policy π_t is the $\epsilon - greedypolicy$ w.r.t $Q^{(t)}$

1.4.2 Explain how to modify the previous scenario if your expected time life increases to 50, the minotaur knows your position and has a probability of moving towards you equal to 35%, and you need to pass throw the right corner ot the maze to take the keys to open the exit door.

Three modifications we should add to the previous exercise in order to address the new problem.

Firstly, regarding the increase of the expected value in the geometrical distribution we should modify the λ discounted factor to 49/50 due to equation 7. This modification implies that the time horizon will be a little bit longer resulting in a higher probability of exiting the maze. It is also important to update the probability parameter of the `numpy.random.geometric` python function to 1/50.

Secondly, the Minotaur is awarded of our position and can move towards our direction with a 30% probability. To simulate this action, we would compute the Manhattan distance between the player and the Minotaur form all the adjacent boxes of the Minotaur and we will modify the transition probabilities based on this number. Previously, the transition probability $P((x_t, y_t, k_t, l_t)|(x_{t-1}, y_{t-1}, k_{t-1}, l_{t-1}), a_p)$ was multiplied by $1/n$ where n was the number of possible actions the Minotaur was allowed to do. Now, this probability will be multiplied be $1/3n$ where *n* is the number of closer states to the current position of the player $[min\ (ManhattamDist\ (x_{minotaur,t+1}, y_{minotaur,t+1}), (y_{player,t}, y_{player,t})]$, and by $2/3n$ when it is moving to another position, being *n* the number of possible actions to perform.

Besides, the player needs to go to the right corner position of the maze in order to take the keys and be able to open the exit door. In order to achieve this we need to modify the states and the reward function in a way that arriving to the end doing the shortest path while avoiding the Minotaur without keys presents a higher penalty than arriving to the final position, avoiding the Minotaur and having the keys. For that reason, we thought of duplicating the number of states by including a binary variable to monitor the keys presence: $S = (x, y, k, l, p)$ where *p = keys preset / keys not present*, and the reward function by including a penalty of arriving to the end position without the keys equal to -30, and to 0 with the keys. After some trials, we also decided to include a reward of +10 to arrive to the box where the keys are placed and also to exit the maze with them.

Finally, to improve the performance of the two consecutive algorithms a time horizon equal to 200 has been set in the training process. If this parameter was not included, the player was prioritizing to run from the minotaur instead of arriving to the goal position.

1.4.3 Q-learning with ϵ-greedy policy.

Q-learning algorithm was implemented following the algorithm indicated in [1]. To balance exploration and exploitation, an ϵ-greedy policy was implemented. In Q-learning with ϵ-greedy policy, the agent chooses the action to take at every step either in a randomized way or according to the policy learned until then, with a probability epsilon. After that, the agent observes the reward and the next step and the Q value of the combination of state and action is updated. This is repeated until a terminal state is reached. Finally, this learning process is done for a number of episodes.

1.4.4 Solve the Problem for 2 different values of the exploration parameter epsilon. Show the convergence of the algorithm with a decaying step size alpha according to the number of times that a pair (s,a) has been visited. Discuss whether a proper initialization of the Q-values may affect convergence speed.

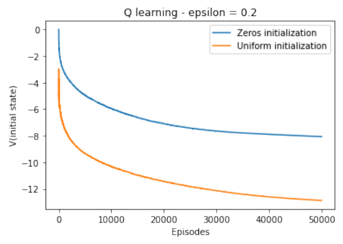
It is clear from figure 3 that the algorithm doesn't really converge in 50000 episodes, as it still shows a tendency to go down. For the algorithm to converge, it needs to go through every pair of states (s,a) an infinite amount of times. As we are repeating the training for a finite number of episodes, the algorithm did not visit all the possible pair of states and actions, consequently,it did not have

Algorithm 1 ϵ-greedy Q-learning

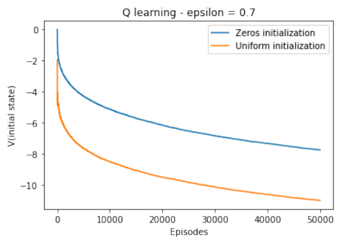
```
Require: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
  Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$, uniformly at random except if $Q(terminal, \cdot) = 0$
  for each episode do
      $S \leftarrow initialstate$
      while S is not terminal do
          Choose A random with probability $\epsilon$ and derived from Q with probability $1 - \epsilon$ ($\epsilon$-greedy
  policy)
          Take action A, observe R and S'
          $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
          $S \leftarrow S'$
      end while
  end for
```

time to converge to the optimal policy as it did not accomplish the convergence's conditions. The convergence that we can observe in graph 3 corresponds to the decay of the learning rate alpha.

The algorithm was tested with 2 different initialization of Q. It is clear that a random uniform initialization allows faster convergence. With an epsilon-soft policy, we follow the best action described by Q with some probability. If all the actions of a state that hasn't been visited yet give the same Q value, the action chosen will always be the first one (stay in this case) reducing the exploration of the algorithm. This is avoided with a random initialization. We can also conclude that a smaller epsilon results faster convergence, since it reduces the randomness of the algorithm.



(a)



(b)

Figure 3: Value function of the initial state over the episodes with different initialisations.

1.4.5 Show the convergence of the Q-learning algorithm for 2 different step sizes $1/n(s, a)^\alpha$ with $\alpha \in (0.5, 1]$.

Observing figure 4, it can be seen that a higher alpha results in a higher decay of the step size, leading to almost no update of the Q values from epoch 5000. In contrast, lower alpha allowes convergence closer to the optimal value, but needs more episodes.

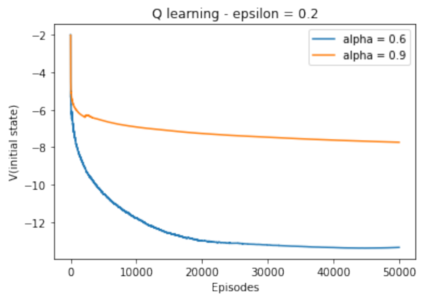


Figure 4: Value function of the initial state over the episodes with different step size decaying factors.

1.4.6 SARSA

SARSA algorithm was also implemented following the algorithm indicated in [1]. The algorithm is really similar to Q-learning, the only difference is that in every step, the action produced in the next state is also taken into account. Then, the learnt Q values also consider the exploration phase (with epsilon-greedy).

Algorithm 2 SARSA

```
Require: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
  Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$, uniformly at random except if $Q(terminal, \cdot) = 0$
  for each episode do
      $S \leftarrow initialstate$
      Choose A from S using policy derived from Q ($\epsilon$-greedy policy)
      while S is not terminal do
          Take action A, observe R, S'.
          Choose A' from S' using policy derived from Q ($\epsilon$-greedy policy)
          $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
          $S \leftarrow S'$
          $A \leftarrow A'$
      end while
  end for
```

1.4.7 Solve the Problem for $\epsilon = 0.1$ and $\epsilon = 0.2$. Show the convergence of the algorithm with a decaying step size according to the number of times that a pair (s,a) has been visited. Discuss whether a proper initialization of the Q-values may affect convergence speed.

Figure 5 shows a clear difference on the value function for different epsilon. Higher exploration parameters introduce more randomness in the Q values - which take into account the exploration phase - and results in higher costs.

The initialization of the Q-values in SARSA play a similar role as in Q-learning (see section 1.4.4). For this reason, it was initialized following random uniform distribution.

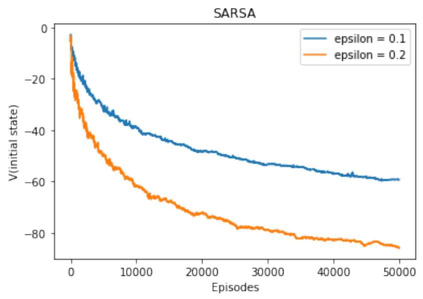


Figure 5: Value function of the initial state over the episodes with different exploration factors.

1.4.8 Consider the case where the exploration parameter ϵ decreases in each episode (for example, in episode k = 1,... choose $\epsilon_k = 1/k^\delta$ with $\delta \in (0.5, 1]$). Does convergence improve? Is it better to have $\alpha > \delta$, or the opposite?

Decaying the exploration parameter furthers the convergence of the algorithm. Decaying the exploration parameter reduces the randomness in the last updates of the value function. If the decay of epsilon is faster than the step size, exploitation will have more weight in the resulting value function. This can be seen in figure 6, as higher decay parameter shows a less noisy value function that converges faster.

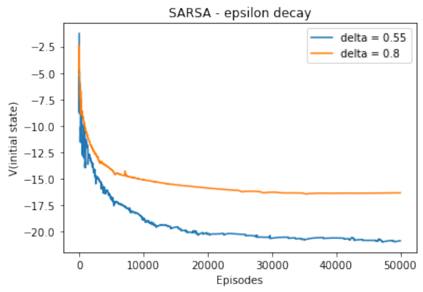


Figure 6: Value function of the initial state over the episodes with different exploration decaying factors. Epsilon is initially set at 0.66.

1.5 Estimate the probability of leaving the maze using (1) a policy computed through Q-learning, and (2) another policy computed using SARSA. Are the probabilities close to the Q-value of the initial state (for the respective learning method)? If so/not, explain why.

The probability of leaving the maze, computed after 10000 simulation, is 42% and 42.07% for q-learning and SARSA policies respectively. The mean time that it takes for the player to exit the maze is around 45 steps. The policies were also checked in simulations were the influence of the venom is not taken into account. In those cases, the exit probability is about 100% in both cases. This suggests that most of the times, if the player doesn't get out of the maze is because the venom kills him before having the chance to reach the exit (it needs at least 30 timesteps to get the key and get out).

Probabilities obtained are not close to the Q-values of the initial state. That is because of the way that the rewards have been designed. If the rewards were to be set as an indicator function such as $f(x) = \mathbb{1}\{x \in A\}$, then by definition we would have $\mathbb{E}[f(x)] = \mathbb{P}(\{x \in A\})$. Since the value function is the expected reward achieved under a policy, the probability of exiting the maze and the Q-values of the initial state would be the same.

2 Problem 2: RL with Linear Function approximators

2.1 Describe the training process: for how many episodes did you train, which values of the parameter did you use, a short description of the algorithm and the Fourier basis and if you used any SGD modification.

In this section we have applied linear function approximation to a RL problem in order to estimate a Q function. The environment consists on a car that can proceed with three different actions: go left, stay and go right. The final goal is to choose an action in a specific position in order to arrive to the end of the road in the lowest number of movements, taking into account that each action has a negative reward equal to minus one.

Despite the previous exercise, this problem is set up in a continuous domain, therefore, linear function approximation, using the Fourier Basis to reference the vectors, has been used to approximate the Q function and deduce a policy from it.

This section also proposes the use of eligibility traces, which work as a memory vector $z \in \mathbb{R}^d$ and have the same dimension as $w \in \mathbb{R}^d$ (weights for the linear approximation). The idea is that when a weight component is updated the eligibility trace is bumped up and starts decaying with a trace decay parameter $\lambda \in [0, 1]$. Thanks to that, we can keep track and give a higher credit to those states that have been visited and have contributed to achieved the goal.

In this case, we have trained the model for 400 episodes with a trace decay (λ) equal to 0.1 and a learning rate (α) equal to 0.2. Moreover, we have modified the learning rate α to decay depending on the rewards achieved, we have add momentum to the velocity tensor to avoid oscillations, we have normalized the alpha (α) value for each η Fourier base and finally, the clipping of the eligibility vector has been applied to improve the performance of the algorithm.

2.2 Do the following analysis of the training process and the optimal policy.

2.2.1 Plot a figure showing how the episodic reward changes across episodes during training.

Figures 7 shows the performance of the training process across episodes. From these results we can conclude that the first episode training is crucial to arrive to a good reward value, and that single episodes present oscillations but their average (windows of ten samples) is quite stable.

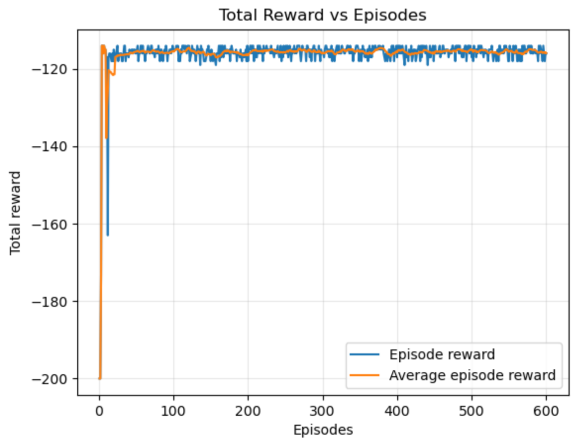
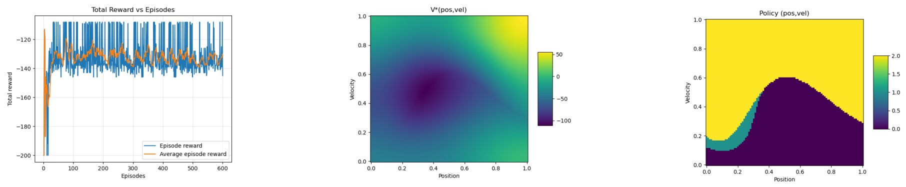


Figure 7: Training process with eligibility trace = 0.1, decay alpha initial value = 0.2 and 400 episodes

2.2.2 Plot a 3D plot of the Optimal value function and policy.

As mentioned in the beginning of the section, this problem presents a continuous environment, therefore we use a linear approximation technique referenced by the Fourier basis to estimate the value function `max(Q)` and the optimal policy. Figures 8 show a 3D (heat plot) of the optimal value function and policy in the continuous space. From these results we can conclude that with low velocities the action to perform should be going to the left, while with high velocities the action should be going to the right. On the other hand, when the velocities are in the middle, the position takes more importance.

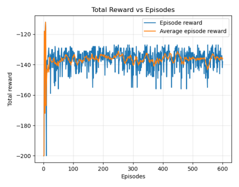


(a) Training process (b) Optimal Value Function (c) Optimal Policy

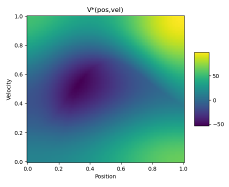
Figure 8: Results for episodes = 600, eligibility trace = 0.1 and alpha initial values = 0.2 with $\eta = [0, 0]$ in the basis

2.2.3 Does it make a difference including $\eta = [0, 0]$ in the basis?

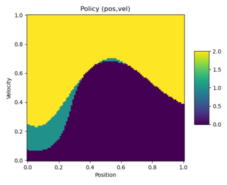
Including $\eta = [0, 0]$ in the Fourier basis gives us the offset of the function, due to $\phi_i(s) = cos(\pi\eta_i^T s) = cos(\pi[0, 0]^T s) = cos(0) = 1$. As figure 9 shows, the final policy is almost equal while not including vs including the $\eta = [0, 0]$ base (the yellow region, which corresponds to right action, is a bit smaller while the green area, which corresponds to stay action, takes a bit more presence), however because of this offset the average reward increases a little.



(a) Training process



(b) Optimal Value Function



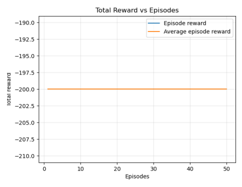
(c) Optimal Policy

Figure 9: Results for episodes = 600, eligibility trace = 0.1 and alpha initial values = 0.2 without $\eta = [0, 0]$ in the basis

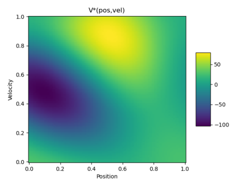
2.2.4 Compare your results with an agent that takes random actions.

In the following section we have compared the results achieve by an agent who behaves complete random (CR), medium random (MR) and not random at all (NR). The first agent selects the next action to perform randomly from a uniform distribution, the second agent has a probability equal to 50% to select the next action randomly or following the policy while the last one only behaves based on the policy.

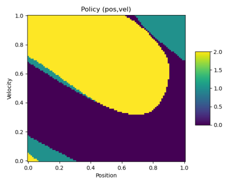
As figure 10 shows, the CR and MR agents do not arrive to the final position at any time, due to all the rewards are equal to -200 during the entire training. As expected, the policy gets more similar to the NR agent policy while decreasing the probability of behaving totally random, due to it is expected that part of the actions follow the policy.



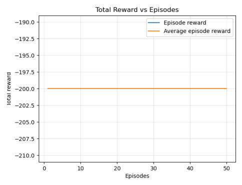
(a) Training process CR



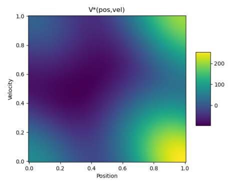
(b) Optimal Value Function CR



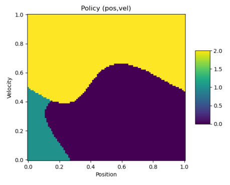
(c) Optimal Policy CR



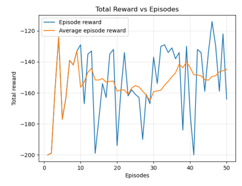
(d) Training process MR



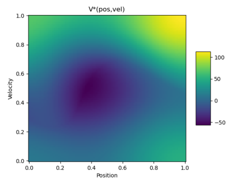
(e) Optimal Value Function MR



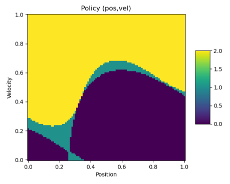
(f) Optimal Policy MR



(g) Training process NR



(h) Optimal Value Function NR

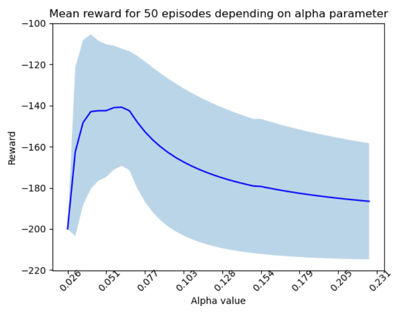


(i) Optimal Policy NR

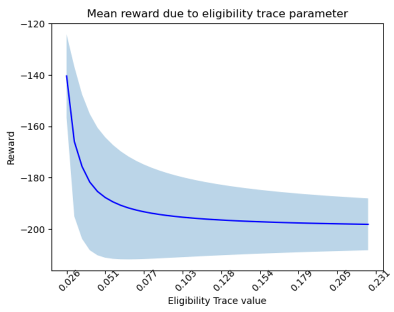
Figure 10: Results for episodes = 600, eligibility trace = 0.1 and alpha initial values = 0.2 without $\eta = [0, 0]$ in the basis

2.3 Compute the average total reward of a policy in function of the learning rate and the eligibility trace. Comment the results.

As figure 11 shows, increasing the learning rate and the trace decay decrease the mean value of the rewards. It seems like a high learning rate can cause the problem to converge to an undesired solution, while a high trace decay quickly decreases the trace parameters, consequently the states are remembered for a shorter time.



(a) Value Function with respect to learning rate

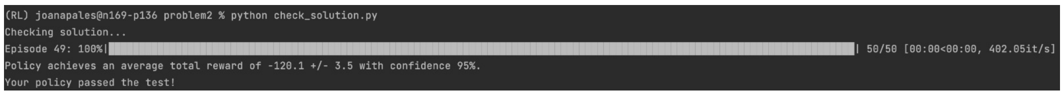


(b) Value Function with respect to trace decay

Figure 11: Results for 100 episodes training

2.4 Evaluate the performance of your agent with the *check solution.py* file.

Figure 12 shows a reward of -120 while using the specification mention in the first section of problem 2.



```
(RL) joanapales@n169-p136 problem2 % python check_solution.py
Checking solution...
Episode 49: 100%|██████████████████████████████████████████████████| 50/50 [00:00<00:00, 402.05it/s]
Policy achieves an average total reward of -120.1 +/- 3.5 with confidence 95%.
Your policy passed the test!
```

Figure 12: Results for 100 episodes training

References

[1] P. R. Montague. Reinforcement learning: an introduction, by sutton, rs and barto, ag. *Trends in cognitive sciences*, 3(9):360, 1999.